Q1)

-final Weights = [0.99662832 0.00135793]
-No. of iterations = 55
-Learning Rate = 0.2
-stopping criteria: Error difference in subsequent iteration less than e-11

Normalizing the data resulted in faster convergence because:
1) lesser iteration required for a given learning rate.
2) Using a higher learning rate was possible, as compared to un-normalized data

Figure1, shows the plot of data-set and the linear regression line on top of it.

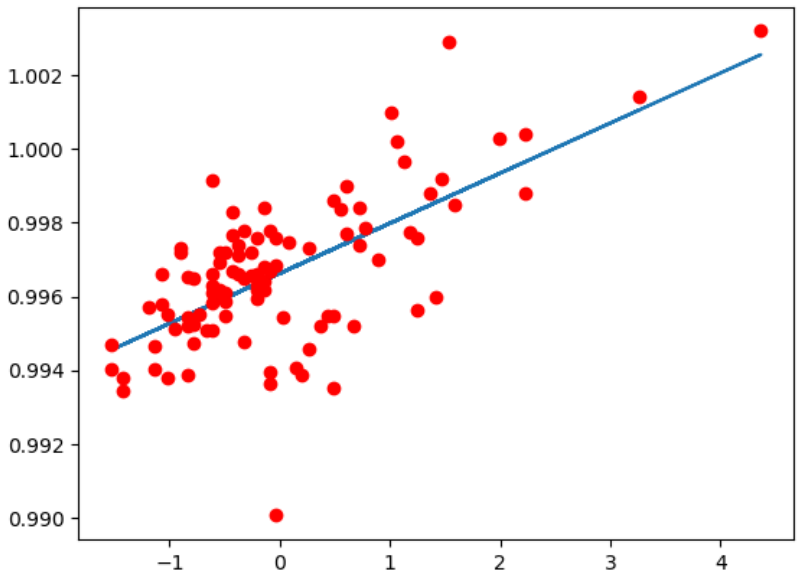

Figure1: linear Regression fit

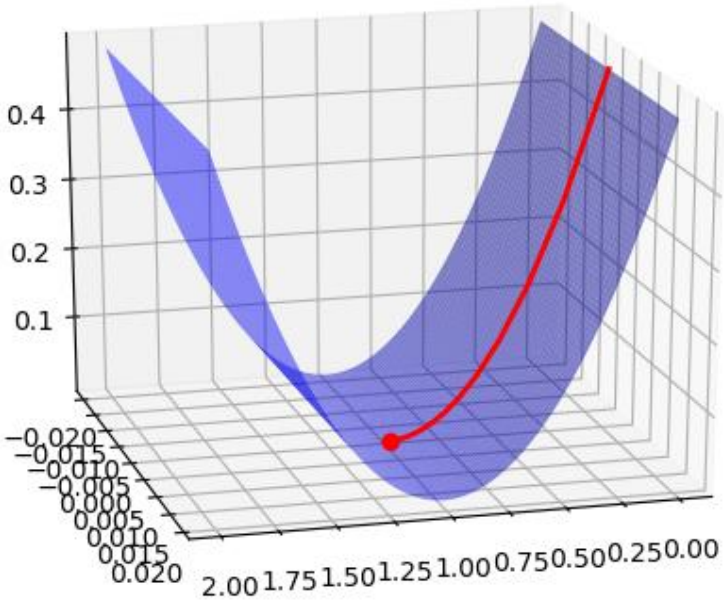

Figure2: Reduction in error with each iteration

In figure 2, red curve indicates reduction of the mean-square error with each iteration. The corresponding values of the red curve indicates weight values.

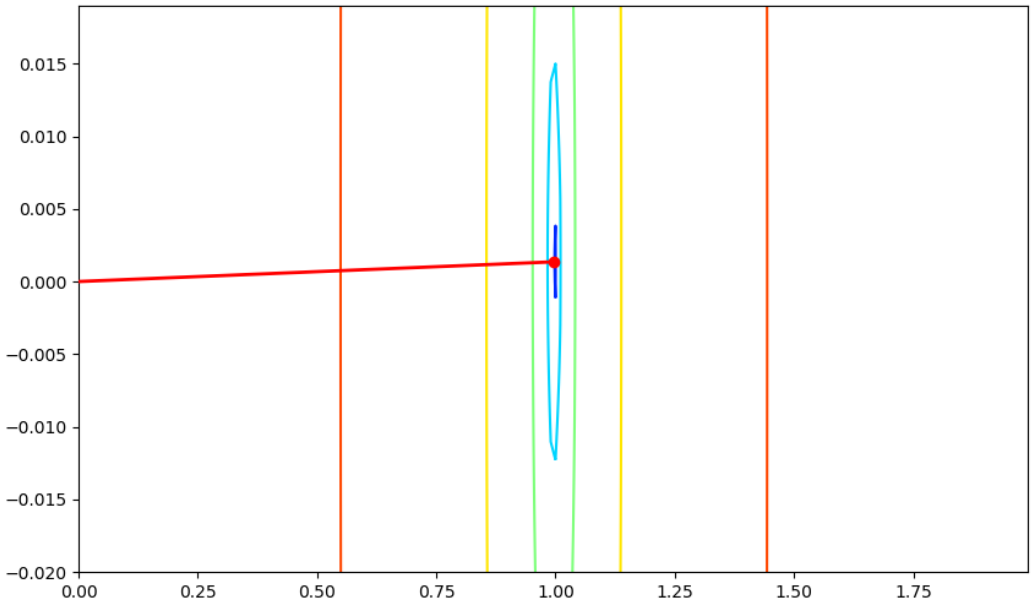

Figure3: Changes in linear regression weights and error contours

The red line in Figure 3 shows the changes in the weights and the path it travels with each iteration.

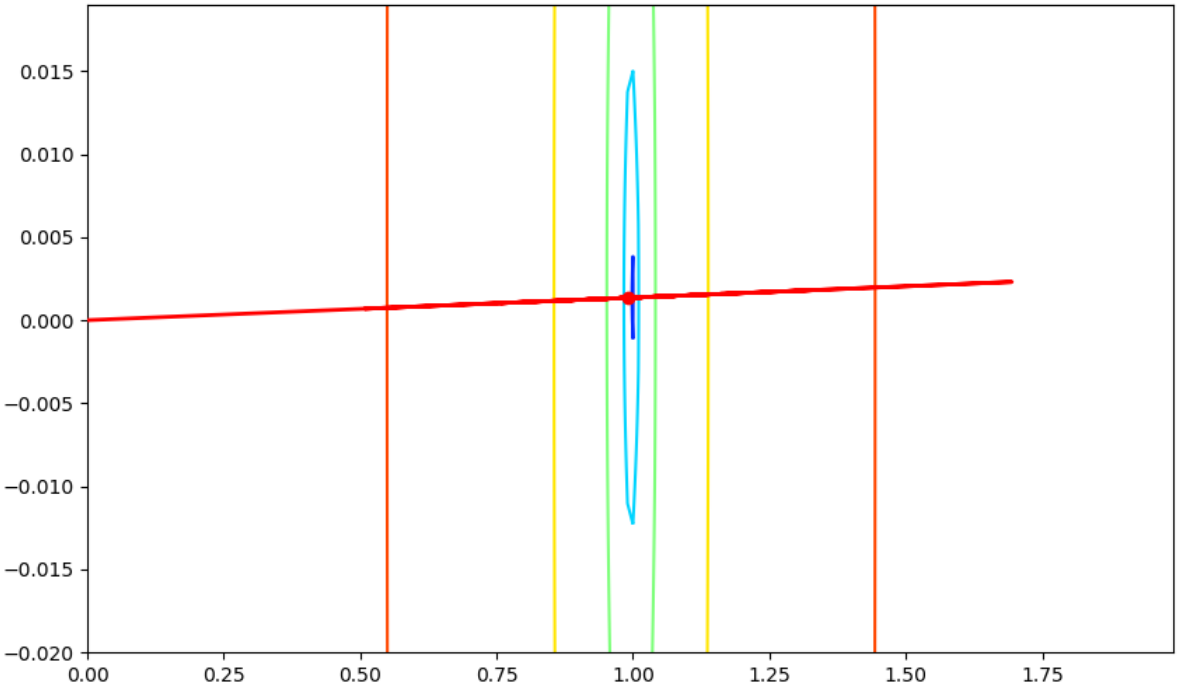

Figure 4: Overshoot with high learning rate = 1.7

We plot contours with various learning_rates(n)=(0.1, 0.5, 0.9, 1.3, 1.7,2.1,2.5).

As n increases the gradient descent converges in fewer iteration because we make bigger updates to parameters per iteration.

However, with n=1.3 and higher, the iteration starts to increase. This is because the big updates make parameters jump to other side of minima. Overshoots start and the weights oscillates around the optimum before converging, as oscillation dampens.

For n>2.1 overflow occurs. The updates are so big that divergence starts.

Q2) Locally weighted linear regression fits piecewise linear approximation to a non-linear dataset. The datapoints that are closer to the point of approximation has higher weights, as compared to the datapoints that are further away.

consider a parametric model in which we explicitly model the variation of the function at $\mathbf{x}_0$, but emphasize nearby points more heavily than those far away. During training, we can do that by weighting the training examples:

$$\theta^*(\mathbf{x}_0) = \sum_{i=1}^{n} L(y_i, f(\mathbf{x}_i;\theta))K(D(\mathbf{x}_i - \mathbf{x}_0; h)/h)$$

Note that here we use a more general notation for the kernel, referring to an arbitrary distance function D.

For squared loss, and linear model, we get an analytical solution as follows. First, we shift the $\mathbf{x}$s by subtracting the query $\mathbf{x}_0$ from each $\mathbf{x}_i$

We set up an $n \times d+1$ matrix $\mathbf{X}$ by placing the shifted $\mathbf{x}_i^T$ in its rows. We denote

$$w_i \triangleq \sqrt{K(\mathbf{x}_i, \mathbf{x}_0)}$$

and

$$\mathbf{W} \triangleq \begin{bmatrix} w_1 & 0 & \dots & 0 \\ 0 & w_2 & \dots & 0 \\ & \ddots & & \\ 0 & 0 & \dots & w_n \end{bmatrix}$$

We then reweight the original data, both $\mathbf{x}$ and y: $\mathbf{z}_i \triangleq w_i\mathbf{x}_i$, so that $\mathbf{Z} = \mathbf{WX}$, and $v_i \triangleq w_i y_i$, so that $\mathbf{v} = \mathbf{Wy}$. Now, we have

$$\begin{aligned} & \theta(\mathbf{x}_0) = \sum_{i=1}^{n} \left(y_i - \mathbf{x}_i^T\theta\right)^2 K(\mathbf{x}_i, \mathbf{x}_0) \\ \Rightarrow \quad & (\mathbf{Z}^T\mathbf{Z})\theta = \mathbf{Z}^T\mathbf{v} \\ \Rightarrow \quad & \theta^*(\mathbf{x}_0) = (\mathbf{Z}^T\mathbf{Z})^{-1}\mathbf{Z}^T\mathbf{v} \end{aligned}$$

The normal equation implementation of locally weighted liner regression (line 28-31 in WeiLinReg.py) is:

```
def weighted_lin_reg_normal_eqn_param(X, Y, W):
  X_ = W@X
  Y_ = W@Y
  return inv(np.transpose(X_)@(X_)) @ np.transpose(X_) @ Y_
```

Figure 1 shows the plot. The red dots represent the dataset. The blue line is the linear regression line fit to the dataset. The green lines are locally weighted linear regression lines at different points of the data-set (-1.0, 3.0, 6.0, 11.0)

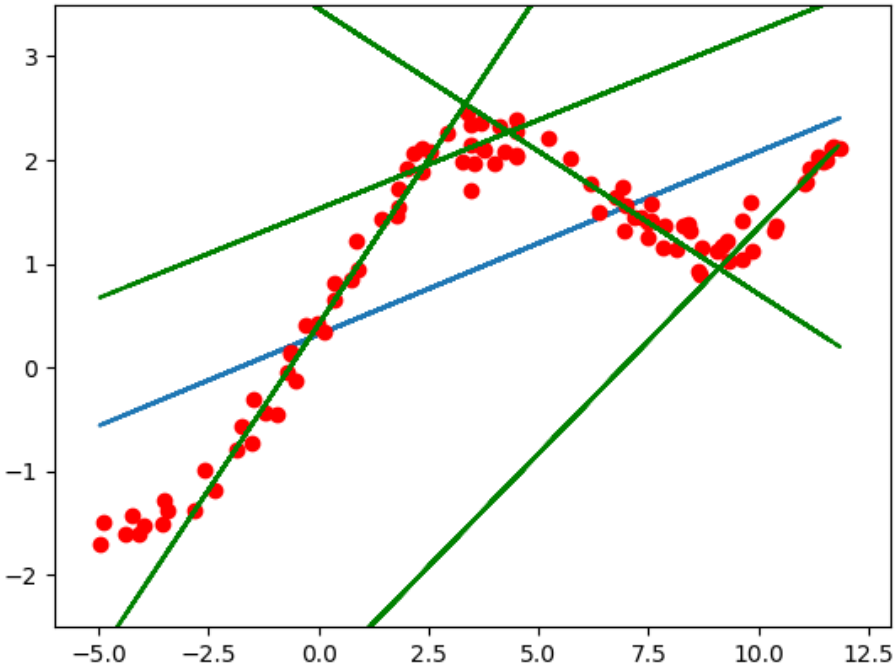

Figure 1: t=0.8

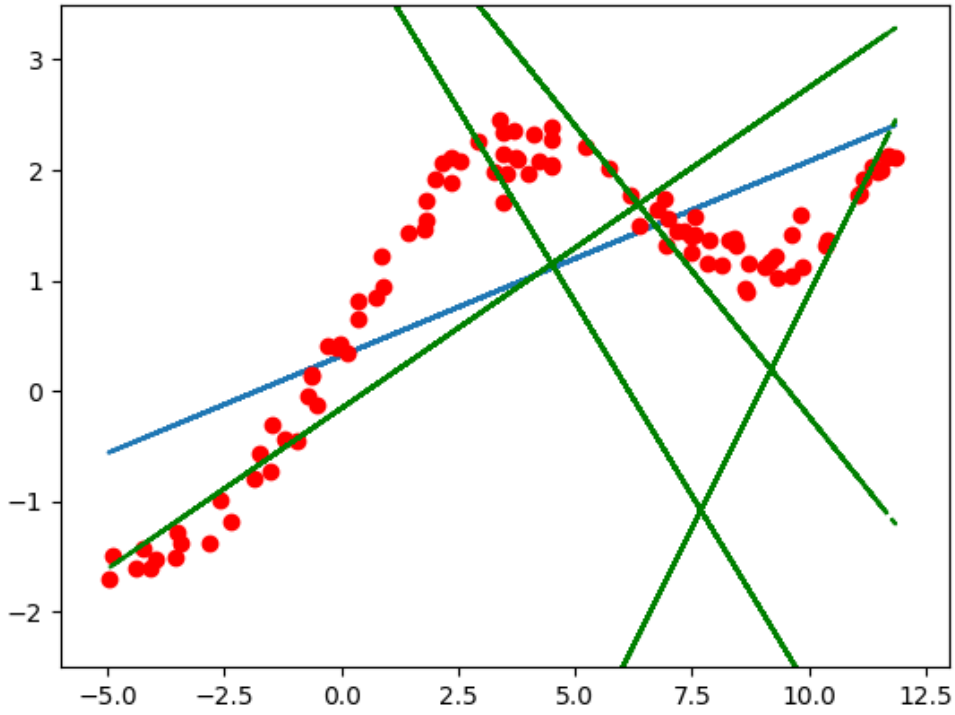

Figure 2: t=0.1

As can be seen in figure2, the green line closely follows only the points in the vicinity, as the weight decays very fast. If there are multiple points near the point of approximation, this may create a problem (eg line approximated at x=3). This problem is similar to overfit.

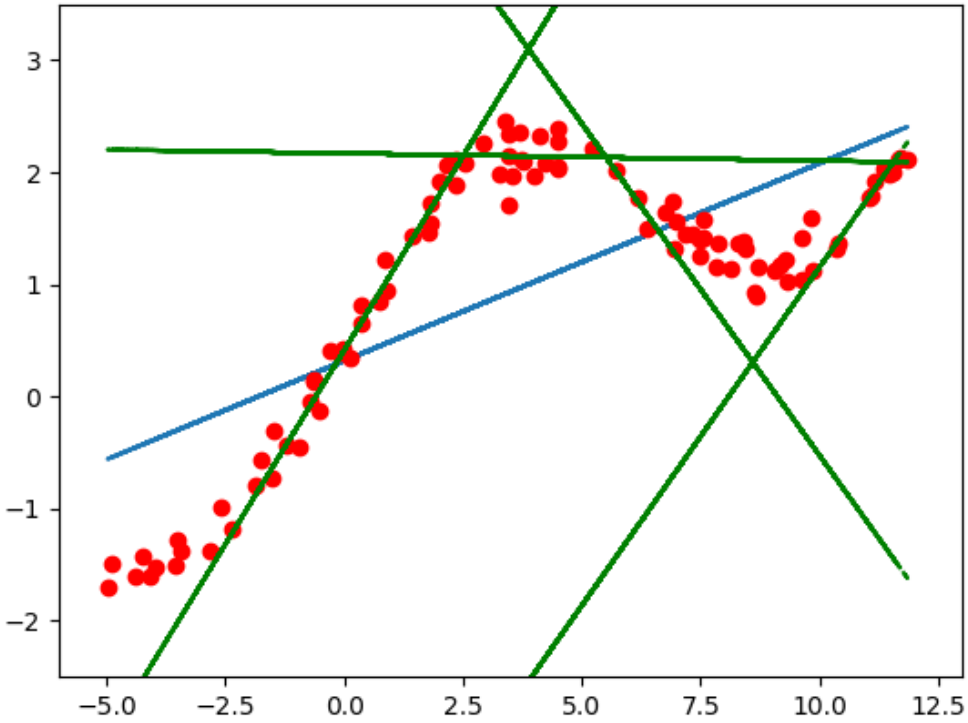

Figure 3: t=0.3

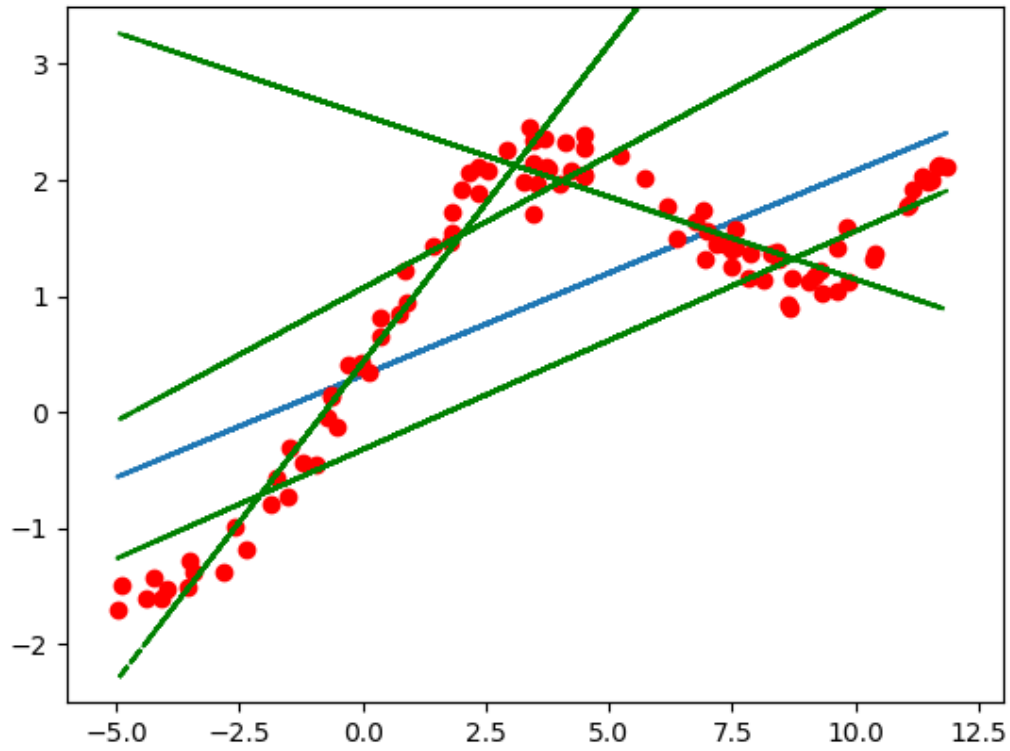

Figure 4: t= 2

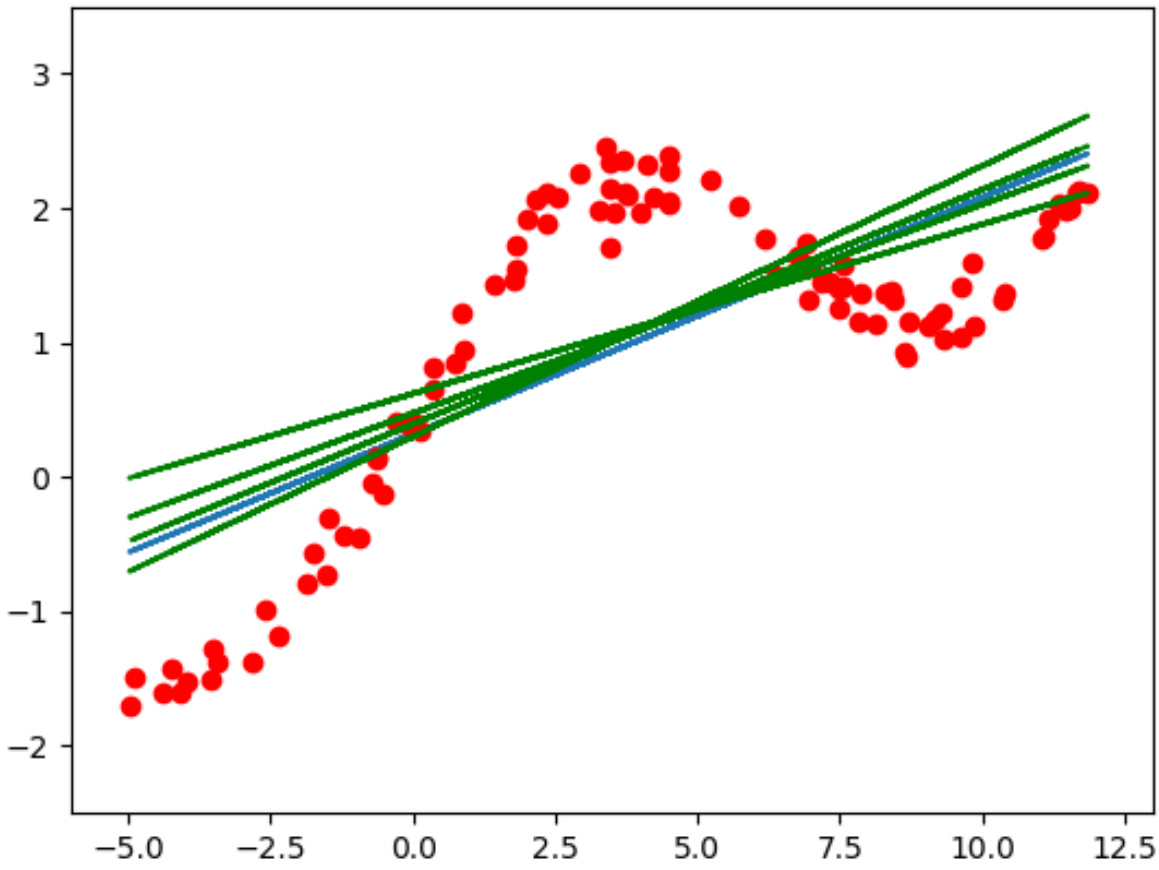

Figure 5: t=10

In this extreme case of t=10, the weights do not decay fast and all the points are utilized for the linear approximation of the dataset. So, these lines very nearly matches the linear regression line. This problem is underfit.

Q3)
theta= [0.55952206 1.93247687 -1.99558285]
iteration= 8

The log likelihood function for logistic regression is:

$$\ell(\theta) = \sum_{i=1}^{n} y_i \log(h_\theta(x_i)) + (1 - y_i) \log(1 - h_\theta(x_i))$$

By substituting **The Hessian** into the **Newton's Method** update step, we are left with:

$$\theta_{n+1} = \theta_n + H^{-1}_{\ell(\hat{\theta})} \nabla \ell(\theta)$$

If the input X is 1-D, the corresponding equations are:

The **gradient** of $\ell(\theta)$ is:

$$\nabla \ell = \begin{pmatrix} \sum_{i=1}^{n} (y_i - h_\theta(x_i)) x_i \\ \sum_{i=1}^{n} (y_i - h_\theta(x_i)) \end{pmatrix}$$

While the **hessian** of $\ell(\theta)$ is:

$$H_{\ell(\hat{\theta})} = \begin{bmatrix} \sum_{i=1}^{n} h_\theta(x_i)(1 - h_\theta(x_i))\theta_1 \theta_1, & \sum_{i=1}^{n} h_\theta(x_i)(1 - h_\theta(x_i))\theta_1 \\ \sum_{i=1}^{n} h_\theta(x_i)(1 - h_\theta(x_i))\theta_1, & \sum_{i=1}^{n} h_\theta(x_i)(1 - h_\theta(x_i)) \end{bmatrix}$$

Where $h_\theta(x) = \frac{1}{1+e^{-z}}$ and $z = \theta_1 x + \theta_2$.

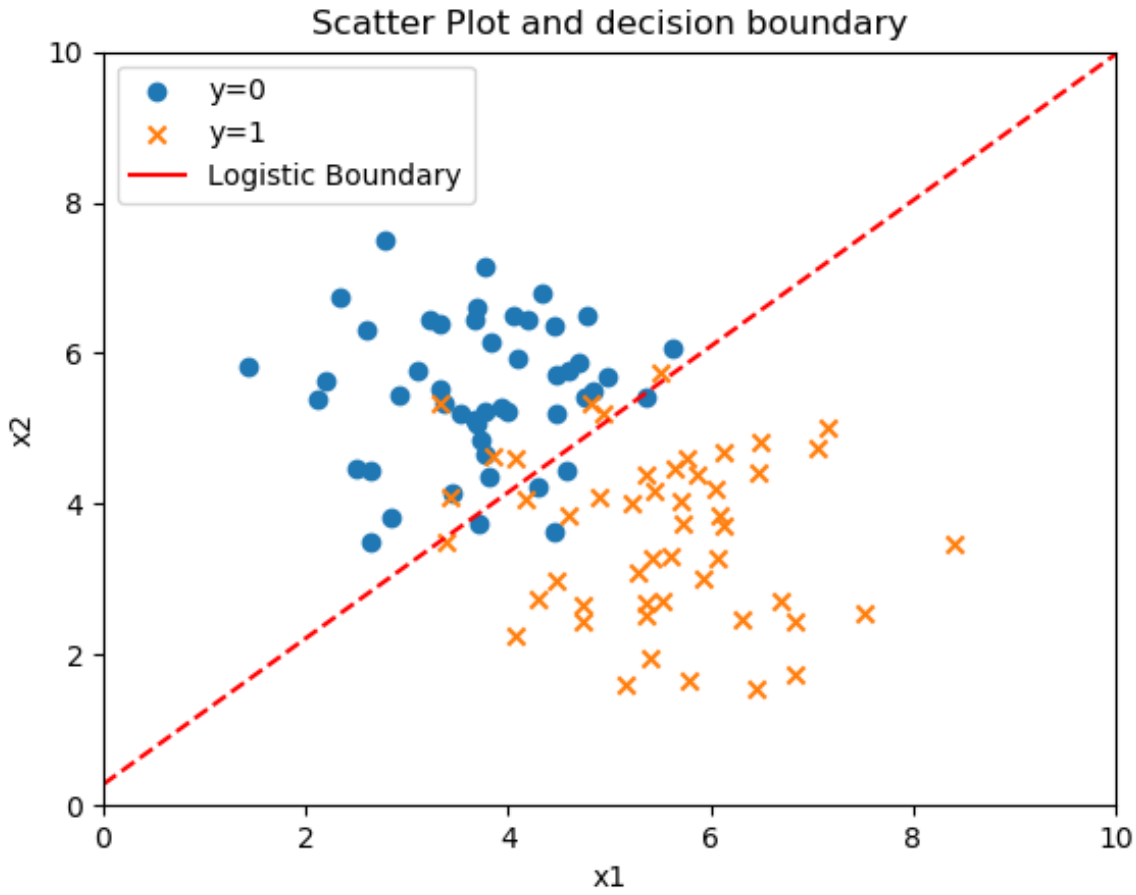

figure1: Logistic regression

Figure1 shows the data and the logistic regression boundary that classifies the data.

Q4)
-mean Alaska = [98.38 429.66]
-mean Canada = [137.46 366.62]
-covariance over whole dataset =
[[669.2936 -642.6488]
 [-642.6488 2116.7604]]
-covariance of Alaska = [[255.3956 -184.3308]
 [-184.3308 1371.1044]]
-covariance of Canada = [[319.5684 130.8348]
 [130.8348 875.3956]]

Linear GDA:

The decision boundaries are defined by:

$$\log \hat{\pi}_k - \frac{1}{2}\hat{\mu}_k^T \hat{\mathbf{\Sigma}}^{-1}\hat{\mu}_k + x^T \hat{\mathbf{\Sigma}}^{-1}\hat{\mu}_k = \log \hat{\pi}_\ell - \frac{1}{2}\hat{\mu}_\ell^T \hat{\mathbf{\Sigma}}^{-1}\hat{\mu}_\ell + x^T \hat{\mathbf{\Sigma}}^{-1}\hat{\mu}_\ell$$

Quadratic GDA:

In **quadratic discriminant analysis** we estimate a mean $\hat{\mu}_k$ and a covariance matrix $\hat{\mathbf{\Sigma}}_k$ for each class separately.

Given an input, it is easy to derive an objective function:

$$\delta_k(x) = \log \pi_k - \frac{1}{2}\mu_k^T \mathbf{\Sigma}_k^{-1}\mu_k + x^T \mathbf{\Sigma}_k^{-1}\mu_k - \frac{1}{2}x^T \mathbf{\Sigma}_k^{-1}x - \frac{1}{2}\log|\mathbf{\Sigma}_k|$$

This objective is now quadratic in x and so are the decision boundaries.

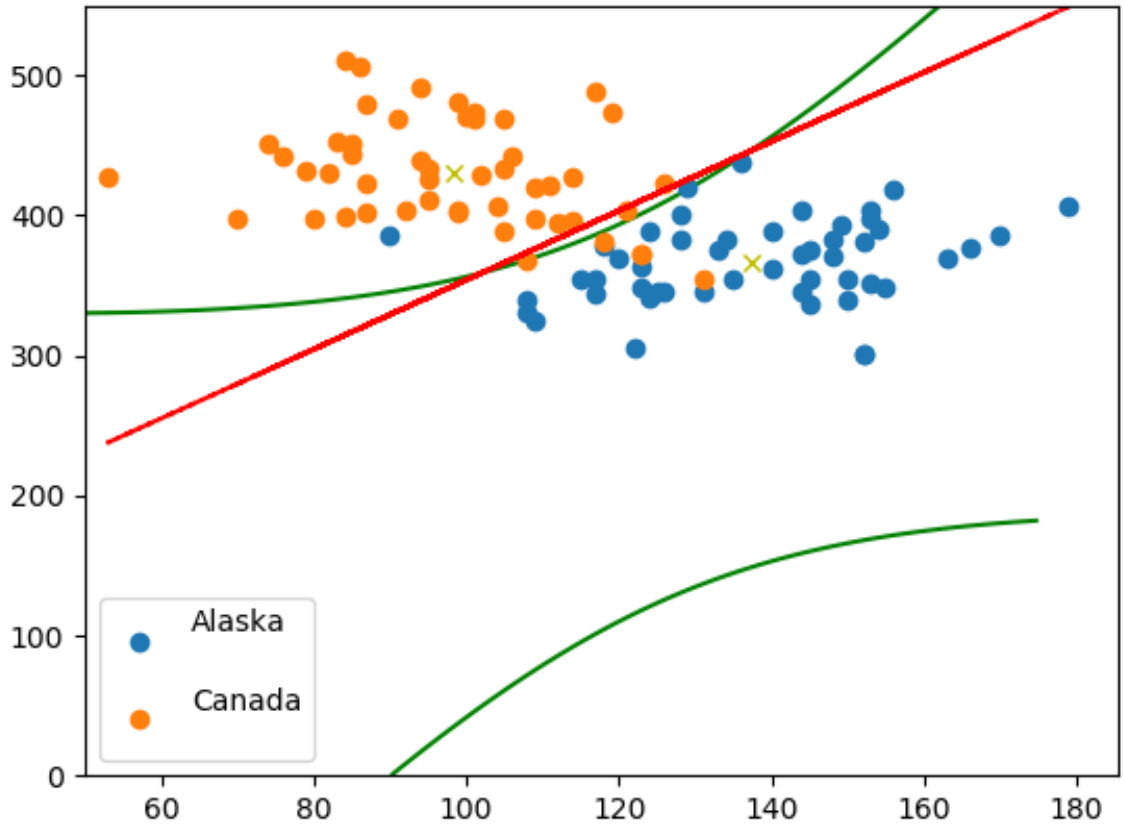

Figure1: GDA Plot

The figure 1, shows the plot of data with the two classes viz. Alaska and Canada, shown in orange and blue color. The linear GDA boundary is plotted in red color and the quadratic GDA boundary is hyperbola, plotted in green color.

The difference in the linear and quadratic boundary:

1) The quadratic boundary classifies a few points very near the decision boundary slightly better than the linear boundary
2) The quadratic boundary classifies the points for low x2 to be Canada, whereas linear boundary classifies them as Alaska